COBB

INSIDE VISUAL dBASE™

Tips & techniques for Visual dBASE and dBASE for Windows

January 1997 • Vol. 4 No. 1
US $8.50

Converting a proprietary application's output to useful information

by Jeff E. Davis

Recently, the owner of a small business hired me to evaluate the company's data processing procedures. My assignment was to determine what, if anything, the office manager could do to reduce the amount of time it was taking each month to close the books and generate a number of reports.

In this article, I'll present how I used Visual dBASE to create a time-saving business solution for this client. Along the way, I'll show you how to accomplish two important aspects of this solution—converting a text file to a dBASE database file and then consolidating key information into the appropriate records.

Replacing computer systems with manual systems with...

This client spent a lot of money to buy a proprietary accounting system. Although the program stores all the right information, it provides only a limited number of reports. Furthermore, the application doesn't offer a custom report option, and the vendor doesn't provide opportunities to purchase or even request additional reports—except with major upgrades.

In order to create reports that the executive officers need to run the business, the office manager had been spending many hours each month manually compiling new reports, based on output from the proprietary application. That's right—the manager printed several canned reports; used a calculator, a pen, and paper to summarize values; and then re-keyed the new reports from scratch.

At this point, you're probably wondering why on earth this client bought the program in the first place—and then kept on using it. In this case (as is the case in many small companies) the business was growing so rapidly that the daily goal was just to keep up with filling orders, leaving little time to improve efficiency in accounting procedures.

Dollars by customer?

Here's the specific problem that my client (a food service provider) faces when using the canned application's standard reports: The sales journal—the report listing all the transactions put on the books each month—includes invoice numbers, but not customer IDs.

Now, suppose you want to know which customer is spending the most money with your company. Sure enough, this canned application provides a report that shows transactions by customer, including invoice numbers. The problem, however, is that the report includes output grouped by customer by product. In order to get a true figure of dollars per customer, the office manager

IN THIS ISSUE

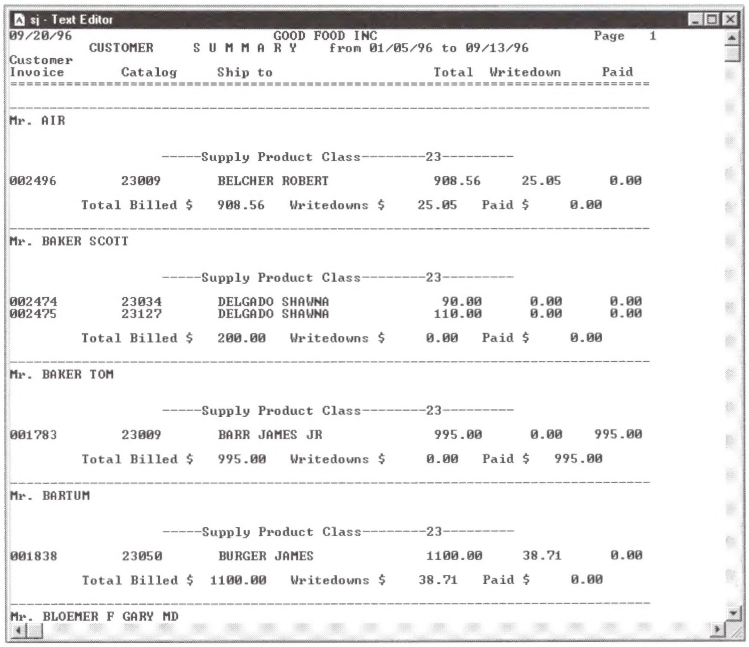
was manually tallying the dollars for each product a customer ordered.

The canned app's saving grace

The proprietary application *did* offer one tool the client didn't know about until I pointed it out—the option of printing the predefined reports to a disk file. I recommended a very simple strategy: Capture the raw data in a disk file. Then, write a Visual dBASE program that uses the disk file to generate the reports the office manager had been creating manually.

After capturing the sales journal report to a disk file named SJ.TXT, I used the command *MODIFY FILE SJ.TXT* to open the file, as shown in Figure A. As you can see, I've grouped the output by customer and within each customer, by supply product class.

Figure A



We'll convert the information in this text file to a database.

Figure B

STEMP - Table Records

Rec	SLINE
1	Mr. AIR
2	002496 23009 BELCHER ROBERT 908.56 25.05 0.00
3	Mr. BAKER SCOTT
4	002474 23034 DELGADO SHAWNA 90.00 0.00 0.00
5	002475 23127 DELGADO SHAWNA 110.00 0.00 0.00
6	Mr. BAKER TOM
7	001783 23009 BARR JAMES JR 995.00 0.00 995.00
8	Mr. BARTUM
9	001838 23050 BURGER JAMES 1100.00 38.71 0.00
10	Mr. BLOEMER F GARY MD
11	002012 23110 BLOEMER MELANIE 695.00 0.00 730.62
12	002162 23058 HUMPHRESS M AMY 85.00 50.94 59.53
13	002161 23110 HUMPHRESS M AMY 695.00 64.00 676.00
14	002509 23110 ELAM STEVE 695.00 0.00 0.00
15	Mr. CABORN DAVID
16	002473 23050 MASON DOROTHY 2400.00 0.00 0.00
17	Mr. CANGEMI PAUL
18	002389 23009 WRIGHT NATHAN 995.00 0.00 0.00
19	002134 23083 REAMS RONNIE 300.00 0.00 300.00
20	001762 23009 MARKS RONALD 995.00 0.00 995.00
21	002189 23082 RACE ROGER J 300.00 0.00 300.00
22	001990 23083 MARCUM JOSH 300.00 0.00 0.00
23	002540 23009 MARCUM JOSH 995.00 0.00 0.00
24	001786 23083 TUTTLE ELVIN 300.00 0.00 0.00
25	001857 23083 GOECKE MIKE 300.00 0.00 240.00
26	002531 23009 GOECKE MIKE 995.00 0.00 0.00
27	Mr. CHOU JEFFREY
28	002491 23001 DOMSCHKE MICHELLE 90.00 0.00 18.00
29	Mr. COLOSIMO, ANGELO
30	001813 23051 GOSSELINK LAURIE 180.00 0.00 0.00
31	001913 23009 WINE DENNIS 995.00 0.00 0.00
32	002546 23009 REMPE STEVE 995.00 0.00 0.00
33	001914 23051 WINE DENNIS 180.00 0.00 0.00
34	Mr. COMPTON RICK

We use a variation of the APPEND command to load this information from a text file into a database.

Creating target databases

Our goal, as mine was, is to create records that include the customer's name on the same line as the invoice number. Then, we can use that database to create a report that automatically subtotals values by customer, which will save the office manager several hours of valuable time each month.

To begin, issue the command *CREATE STEMP* to create a temporary database named STEMP. Define a single character field named *SLINE* with a width of 80. Then, press [Ctrl]W to save this database design. When the Add Or Generate Records dialog box appears, choose Done.

Now, create another database to store the results of this operation. Issue the command *CREATE SALES*, and this time define a single character field named *SALELINE* with a length of 110.

Once you've defined those two databases, issue the command *USE STEMP* to open the first database. Now, if you look closely at Figure A, you'll notice the only lines in the text file that matter are the lines listing the customer names, and the detail lines (the lines listing the invoice number, catalog number, and so on). We know that all the customer names lines begin with *Mr.*, and we know that all the detail lines begin with an invoice number whose first two digits are *00*. Therefore, to load those lines into the STEMP database, issue the command

```
APPE FROM SJ.TXT FOR 'Mr.'$SUBS(SLINE,1,3)
    .OR. '00'$SUBS(SLINE,1,2) TYPE SDF
```

The TYPE SDF parameter tells Visual dBASE to read from a text, or ASCII, file. The FOR clauses determine which lines of text Visual dBase copies into records. Note that you use

the name of the target field—SLINE—when you refer to the incoming text in an expression.

After Visual dBASE appends the appropriate records, issue the *BROWSE* command to verify the results. As **Figure B** shows, our STEMP database now contains all the information we need. The next step is to consolidate the customer names into the detail records.

Listing A shows the program SALELINE, which takes the customer names and the detail records in STEMP and consolidates them into single records in SALES. Just issue the command *MODIFY COMMAND SALELINE* and enter the code. Next, issue the command *DO SALELINE*, and then open the SALES database and issue the *BROWSE* command. When you do, you'll see that the program combined the appropriate customer names and detail lines into single records, as shown in **Figure C**. Now, you can easily use this database to generate a report that summarizes the values associated with each customer.

By the way, since we're using a proportional font in our Visual dBASE browse window, the data in the figure may look as though it doesn't line up properly. However, we used the expression SUBS(SLINE,1,30) to extract the customer name, so the program combined that 30-character string with the detail records to create records with a uniform format.

How SALELINE works

The SALELINE program is fairly straightforward. The key is knowing that the first line of STEMP, our source file, contains the first customer name. After opening the source and target files in different work areas, the command

```
TCUST=SUBS(SLINE,1,30)
```

extracts the first 30 characters from the SLINE field and stores the characters in the TCUST variable. The value 30 is arbitrary, since you assume that no entry in the customer name column will be longer than 30 characters.

After skipping to the first known detail record—the second record in the source file—you're ready to begin a loop to check all the records in the file. If the IF test

```
IF 'Mr'$SUBS(SLINE,1,2)
```

evaluates to true, you know you've hit a new customer name. Then, the command

```
TCUST=SUBS(SLINE,1,30)
```

replaces the last customer's name with the new one. Otherwise, you know you're processing a detail record, and the command

```
TDETAIL=SLINE
```

Listing A: *SALELINE.PRG*

```
* SALELINE
* Take data from STEMP.DBF and make
* a new record within file SALELINE.DBF.

SELE 1
USE STEMP
SELE 2
USE SALES
SELE 1
TCUST=SUBS(SLINE,1,30)
SKIP
DO WHILE .NOT. EOF()
  IF 'Mr'$SUBS(SLINE,1,2)
    TCUST=SUBS(SLINE,1,30)
  ELSE
    TDETAIL=SLINE
    SELE 2
    APPE BLANK
    REPL SALELINE WITH TCUST+TDETAIL
    SELE 1
  ENDIF
SKIP
ENDDO
CLOSE ALL
```

Figure C

SALES - Table Records

Rec	SALELINE
1	Mr. AIR 002496 23009 BELCHER ROBERT 908.56 25.05 0.00
2	Mr. BAKER SCOTT 002474 23034 DELGADO SHAWNA 90.00 0.00 0.00
3	Mr. BAKER SCOTT 002475 23127 DELGADO SHAWNA 110.00 0.00 0.00
4	Mr. BAKER TOM 001783 23009 BARR JAMES JR 995.00 0.00 995.00
5	Mr. BARTUM 001838 23050 BURGER JAMES 1100.00 38.71 0.00
6	Mr. BLOEMER F GARY MD 002012 23110 BLOEMER MELANIE 695.00 0.00 730.62
7	Mr. BLOEMER F GARY MD 002162 23058 HUMPHRESS M AMY 85.00 50.94 59.53
8	Mr. BLOEMER F GARY MD 002161 23110 HUMPHRESS M AMY 695.00 64.00 676.00
9	Mr. BLOEMER F GARY MD 002509 23110 ELAM STEVE 695.00 0.00 0.00
10	Mr. CABORN DAVID 002473 23050 MASON DOROTHY 2400.00 0.00 0.00
11	Mr. CANGEMI PAUL 002389 23009 WRIGHT NATHAN 995.00 0.00 0.00
12	Mr. CANGEMI PAUL 002134 23083 REAMS RONNIE 300.00 0.00 300.00
13	Mr. CANGEMI PAUL 001762 23009 MARKS RONALD 995.00 0.00 995.00
14	Mr. CANGEMI PAUL 002189 23082 RACE ROGER J 300.00 0.00 300.00
15	Mr. CANGEMI PAUL 001990 23083 MARCUM JOSH 300.00 0.00 0.00
16	Mr. CANGEMI PAUL 002540 23009 MARCUM JOSH 995.00 0.00 0.00
17	Mr. CANGEMI PAUL 001786 23083 TUTTLE ELVIN 300.00 0.00 0.00
18	Mr. CANGEMI PAUL 001857 23083 GOECKE MIKE 300.00 0.00 240.00
19	Mr. CANGEMI PAUL 002531 23009 GOECKE MIKE 995.00 0.00 0.00
20	Mr. CHOU JEFFREY 002491 23001 DOMSCHKE MICHELLE 90.00 0.00 18.00
21	Mr. COLOSIMO, ANGELO 001813 23051 GOSSELINK LAURIE 180.00 0.00 0.00
22	Mr. COLOSIMO, ANGELO 001913 23009 WINE DENNIS 995.00 0.00 0.00
23	Mr. COLOSIMO, ANGELO 002546 23009 REMPE STEVE 995.00 0.00 0.00
24	Mr. COLOSIMO, ANGELO 001914 23051 WINE DENNIS 180.00 0.00 0.00
25	Mr. COMPTON RICK 002443 23007 PINKERTON JOHN 22.00 0.00 0.00
26	Mr. COMPTON RICK 002444 23007 COOPER MARILYN 22.00 0.00 44.00
27	Mr. COMPTON RICK 002442 23099 CRONE MIL ED 32.00 0.00 0.00
28	Mr. COMPTON RICK 002561 23099 SMITH DORIS 32.00 0.00 0.00
29	Mr. COMPTON RICK 002343 23099 BLANTON GARY 32.00 0.00 0.00
30	Mr. COMPTON RICK 002329 23114 HAGGERTY JERRY 199.00 0.00 0.00

You'll use this database to generate a report that summarizes values by customer name.

stores the entire 80-character field in the variable TDETAIL. Finally, open the target file, add a blank record, and replace the SALELINE field with a string consisting of the stored customer name and detail record.

The moral of the story

In this article, we demonstrated yet another way in which Visual dBASE can help you create proactive business solutions for your clients. Remember, no matter what your clients think their canned applications can or can't do, all you need is the *output to disk* option to provide the raw data necessary for a Visual dBASE solution. ❖

Jeff Davis is an editor-in-chief for The Cobb Group, a business unit of Ziff-Davis Publishing, and does database consulting for fun and profit. You can write to Jeff at **Jeff_Davis@mm.cobb.ziff.com**.

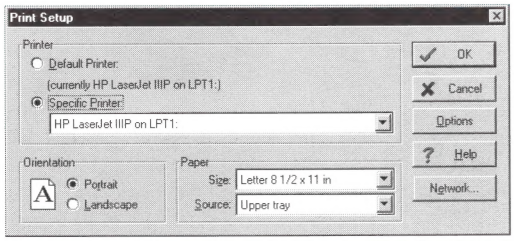
DEVELOPER TIP

Letting your users select a printer

If all your end users relied on the same printer model, you'd have no trouble programming report-printing programs. However, if you install a system on a network, your end users might want to print to any one of several types of printers. This situation poses a programming challenge, because each time you want to print a report, you must allow the user to select a printer to use as the output device.

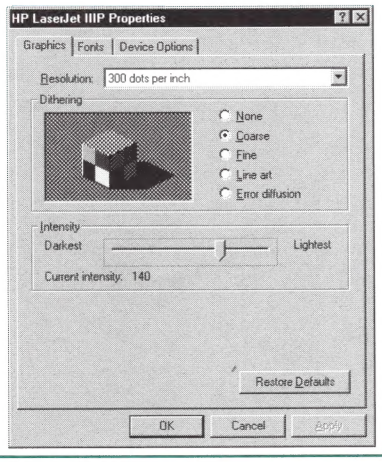
Fortunately, Visual dBASE offers the CHOOSEPRINTER( ) function, which makes it easy for you to let an operator select a printer and a set of print attributes. By using this function instead of a complex series of procedures, you can simplify and standardize your report-processing routines. In addition, end users who are accustomed to a graphical interface will appreciate the Windows "look and feel" of your application.

Figure A



You can display this printer dialog box from within your application by using a single command.

Figure B



You can use this dialog box to quickly customize your printer's properties.

The CHOOSEPRINTER( ) function

You call the CHOOSEPRINTER( ) function with a command in the form

```
return_val = CHOOSEPRINTER( )
```

where *return_val* is the variable that contains the code CHOOSEPRINTER( ) returns to your program. The value of *return_val* will be True if the operator selects the OK button. If the operator presses Cancel, the function will return a value of False. When you call the function, Visual dBASE will display the default Print Setup dialog box, shown in Figure A.

When the dialog box appears, the operator can select the default printer option just by clicking OK, or by pressing [Enter]. The operator can also choose a different printer

by clicking on the Specific Printer radio button and selecting a printer from the list.

Printer options

The Print Setup dialog box lets the operator change the Orientation setting to print either in Portrait or Landscape mode, as well as choose the Paper Size and Source. In the Print Setup dialog box, you'll find a complete Help text, which appears when the user clicks the Help button. Your end users can even customize print options for the selected printer by clicking the Options button. For example, **Figure B** shows the Properties dialog box for the HP LaserJet IIIP printer.

Customizing the Print Setup dialog box's appearance

If you want to customize the title of the Print Setup dialog box, you can do so by passing the CHOOSEPRINTER( ) function an optional string as a parameter in the form

```
return_val = CHOOSEPRINTER("custom_title")
```

Simply replace *custom_title* with the text you want to appear at the top of the dialog box. For example, if you call the function with the command

```
return_val = CHOOSEPRINTER("Choose your
    favorite printer")
```

Visual dBASE will substitute the string of text for the title, displaying a dialog box like the one shown in **Figure C**.

Using the return value

As we mentioned, if the operator presses the Print Setup dialog box's Cancel button, the function returns a value of False. If the operator chooses OK, the function returns a value of True. Once the function returns either value, you can use an IF test to make sure the operator approves the printer selection before your program prints a report. Your program might include the commands

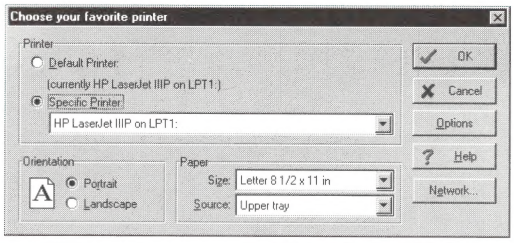
```
IF CHOOSEPRINTER( )
  REPORT FORM myreport TO PRINT ;
  NOCONSOLE
ENDIF
```

The REPORT FORM command will print the report you specify only when the operator properly selects a printer and presses the OK button.

Conclusion

In this article, we showed you how to let end users choose a printer from within your application. When you use the CHOOSEPRINTER( ) function to display a dialog box, you'll save a significant amount of coding time and give your report-processing programs a professional look and feel. ❖

Figure C



You can use the CHOOSEPRINTER( ) function to customize the title of the Print Setup dialog box.

Electronic support with Borland's *Visual dBASE News* and more

If you have an E-mail address, you can subscribe to Borland's Visual dBASE electronic newsletter, *Visual dBASE News*. Each month, Borland publishes up-to-the-minute news about Visual dBASE, including information about new releases and features, power programming techniques, and more.

Visual dBASE News is a read-only Internet mailing list published and distributed to subscribers the first of each month, with occasional special editions. To subscribe, send Internet E-mail to

listserv@borland.com

The body of your message should include the text

```
subscribe visual_dbase YOUR_FIRST_NAME
    YOUR_LAST_NAME
```

Borland also supports transmission of the newsletter to multiple mailboxes for a single user. You can find a subscription form on the World Wide Web at

http://www.borland.com/Connect/interact.htm

If you have any problems receiving your subscription, contact the list owner at **chofmann@borland.com.**

For free technical advice

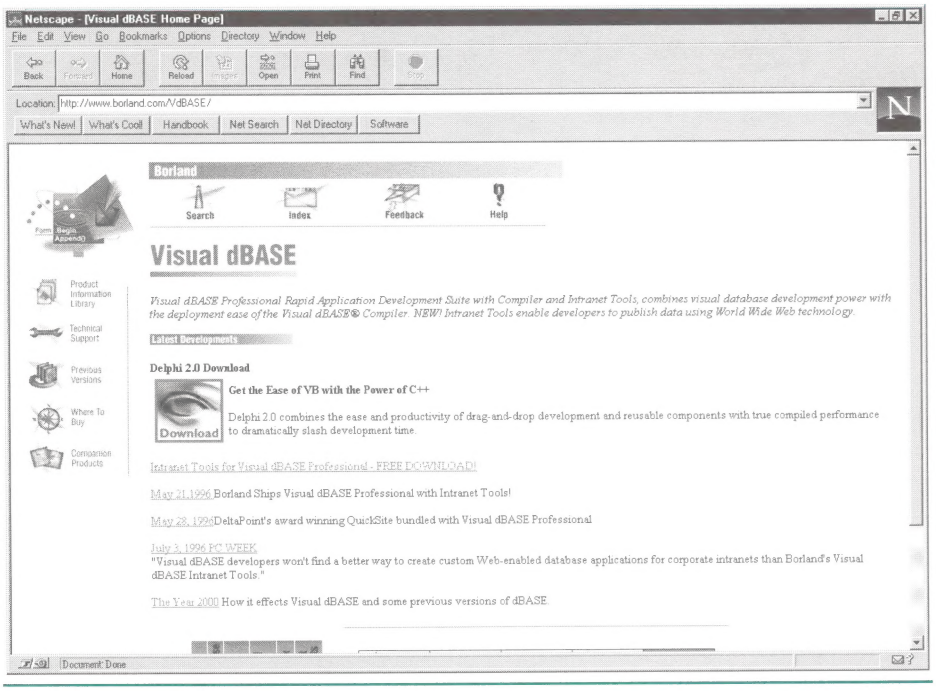
The fastest way to get free online technical support is to post to the CompuServe VDBASE Forum or to the Usenet Newsgroups **comp.databases.visual-dbase** and **comp.databases.xbase.misc**. Representatives from Borland and volunteer product experts monitor the questions; they'll provide quick answers that you can share with other readers. The answer to your particular question may already be waiting for you online! For additional Borland technical support options, see the section *Questions & Comments* on the dBASE Technical Support Web page at

http://www.borland.com/techsupport/VdBASE/

If you'd like to submit product design suggestions to the Visual dBASE Development Team, or submit a contribution to the electronic newsletter, send E-mail to **vdbase@borland.com**.

Net surfers can visit Borland at **www.borland.com**. Then, click on the Visual dBASE tag to visit the Visual dBASE page, shown in **Figure A**. From here, you can locate up-to-the-minute information about new releases, add-ons, and the like. ❖

Figure A



Borland has a Web page dedicated to Visual dBASE.

Using SET( ) to determine the path

In many applications, you want to determine whether a particular directory or subdirectory is currently part of your SET PATH command. Fortunately, the SET( ) function makes it easy to find out exactly what directories or subdirectories are currently defined in your PATH. In this article, we'll show you how to use this function in your applications.

Reviewing SET PATH TO

When you issue a command such as USE or DO from the Command window, Visual dBASE looks for the database or program file in the current working directory. If the program doesn't find the file you specified, an error occurs.

You can change the working directory on an as-needed basis by issuing the SET DIRECTORY TO command followed by the name of the new working directory. However, that approach requires a lot of keystrokes, especially if you're developing a system and working with multiple versions of a file in different directories.

Fortunately, you can use the SET PATH TO command in Visual dBASE much like you use the DOS commands PATH or SET PATH. From the Command window or from within a program, you simply issue a command in the form

```
SET PATH TO directorylist
```

where you replace *directorylist* with a list of directories you want Visual dBASE to search for files. If Visual dBASE doesn't locate a file in the current working directory, it looks in the first directory you named with the SET PATH TO command. Then, the program continues looking through each directory in the list until it finds the file. For instance, you might issue a command like

```
SET PATH TO \JANUARY, \FEBRUARY, \MARCH
```

to tell Visual dBASE to look in those directories for files you refer to in commands such as USE or DO.

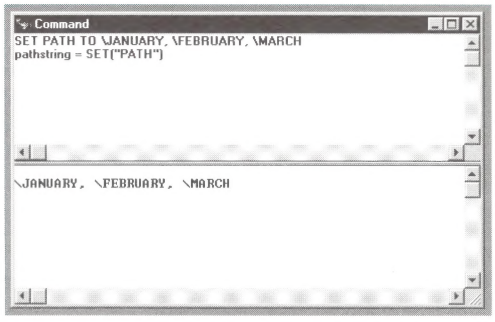
Introducing SET( )

Once you use the SET PATH TO command to define a list of directories for Visual dBASE to use, you can use the SET( ) function to return the list of directories in a string variable. You simply call the SET( ) function in the form

```
pathstring = SET("PATH")
```

When you do, Visual dBASE returns a string that contains the directory names you entered using the SET PATH TO command, as shown in **Figure A**. If no PATH setting exists, the function returns a null string.

Figure A



Use the SET("PATH") function to return the value of the current PATH.

Applying the technique

Now that you know how the SET( ) function works, you're ready to use it in an application. Suppose you maintain 12 months of databases separated into their own directories. You use the SET PATH TO command to place the \JANUARY, \FEBRUARY, and \MARCH directories into the PATH setting.

As you're processing transactions, you want to make sure that your program is working on records for the correct time period. To do so, you need to save in separate variables the month that corresponds to the transaction date and the current PATH setting. For instance, your code might include the command

```
currmonth = CMONTH(transdate)
```

to capture the month associated with the transaction date. Then you'd use the command

```
currpath = SET("PATH")
```

to store the current PATH setting.

Next, you determine whether the string value of currmonth exists as part of the PATH setting in currpath. To do so, you use the substring operator ($) and a simple IF test in the form

```
IF UPPER(currmonth)$currpath
   * Current path is okay.
 ELSE
   * Take corrective action.
ENDIF
```

You may wonder why we used the UPPER( ) function to convert the month's name to all uppercase. We did so because the CMONTH( ) function returns the month name in mixed-case letters, and we entered our pathnames in all uppercase letters when we issued the SET PATH TO command.

Conclusion

In this article, we reviewed how to control where Visual dBASE searches for files, and we showed you how to use the SET( ) function to capture the new PATH setting in a variable. You can use this information in a program to verify that the program is using the correct working directory. ❖

OBJECT-ORIENTED PROGRAMMING

Using codeblocks as properties

In earlier versions of dBASE, when you wanted to execute a block of code, you created a procedure or function and then called it from your program. Your application was often cluttered with lots of short procedures and functions that performed specific repetitive actions.

Visual dBASE and dBASE for Windows provide an important tool that lets you insert lines of code into properties without specifically classifying the code as a procedure or a function. You refer to the line or lines of code as a *codeblock*.

With object-oriented programming, you can create a group of one or more dBASE statements and attach that group to an object's property. This ability saves you the trouble of building, naming, and calling separate procedures or functions.

The basics

When you build a codeblock, you must follow certain formatting rules so Visual dBASE can properly execute the statements within your code grouping. In general, a codeblock takes the form

```
{|parameters|expression_or_statements}
```

where you replace *parameters* with an optional set of parameters delimited by commas and enclosed within the pipe (|) characters. You replace *expression_or_statements* with either an expression or one or more statements, separated by semicolons. In all cases, you enclose the codeblock within the curly braces to differentiate it from other forms of code and data.

Two types of codeblocks

The concept of grouping commands in a codeblock is especially useful when you work with the object's properties. There are two basic types of codeblocks: expression codeblocks and statement codeblocks. When you want to attach a given expression to a property, you use an expression codeblock. When you want to associate a codeblock with a program event, you create a statement codeblock.

The expression codeblock

The term *expression codeblock* refers to a group of code statements that contain dBASE expressions. An expression codeblock resembles a standard dBASE user-defined function (UDF) in that a codeblock returns a value. However, an expression codeblock differs from a UDF in that you don't need to give the block of code a name, store it in a procedure file, or make sure you load it within your application.

An expression codeblock takes the form

```
{|parameters|expression}
```

You replace *parameters* with optional parameters delimited by commas, and you

replace *expression* with the dBASE code that creates the desired expression.

Since you're going to attach this expression codeblock to a property, you can enter the codeblock directly into the property definition itself. For example, suppose you have an order-entry screen that contains a field called ORDER_NO, and you don't want the operator to leave the field blank or enter a negative number. You can implement that constraint by entering an expression codeblock directly into the entry field's VALID event property. The expression codeblock takes the form

```
{ORDER_NO>0}
```

This simple expression codeblock acts as a function and validates the data the operator enters into the ORDER_NO field. The codeblock will return a value of False if the expression is invalid and a value of True if it's correct.

The statement codeblock

As you've seen, an expression codeblock consists of an expression that evaluates to True or False. On the other hand, a *statement codeblock* doesn't necessarily return a value. Instead, a statement codeblock performs an action as a result of an event, such as when the operator presses a particular button in a dialog box.

The statement codeblock contains one or more statements (or commands) that dBASE executes each time an event occurs. The statement codeblock takes the form

```
{|parameters|;statement1; statement2; ...}
```

where you replace *parameters* with optional parameters that you delimit with commas and enclose within vertical lines. You replace *statement1* and *statement2* with the individual dBASE statements you want to execute. You must begin each statement—including the first statement—with a semicolon, even if you're not passing any parameters in the codeblock.

To illustrate, suppose you want to place on a form a button that prints the form. You simply enter a statement codeblock in the appropriate event property of the form, using the command

```
{;this.Print() }
```

This statement codeblock contains one command that tells dBASE to print the form whenever the operator presses a particular key—the event to which the codeblock is attached. Even though the codeblock contains only one command, you must precede that command with a semicolon.

Conclusion

As you learn more about event-driven programming and object-oriented code, you'll find many uses for both expression and statement codeblocks. You can experiment with this technique by assigning a codeblock's parameters to memory variables or associating a codeblock's actions with keyboard and mouse events. ❖

For more information about using codeblocks in a form's event properties, see "Printing a Form," below.

FORM TIP

Printing a form

Suppose you create a screen that contains a single- or multi-file browse window. You locate the record you're looking for, and you want to print it exactly the way it appears onscreen. Unfortunately, in Windows, simply pressing [PrintScreen] doesn't do the job.

The good news is, you don't have to create a special report or build a complex query to get a hard copy of the data you want. In this article, we'll show you how to use the Print( ) function to print an image of the current form that includes all visible objects and data. In addition, we'll show you how to call the Print( ) function by double-clicking the right mouse button.

A single line of code

The biggest advantage to using the Print( ) function to print a form image is that you

can do so by adding only a single line of code to one of your form's Event properties. This single line of code occupies a *codeblock*. A statement codeblock is a nameless group of dBASE commands that you can attach to an object's properties. (For more information about statement codeblocks, please refer to "Using Codeblocks as Properties," on page 8.)

Adding a codeblock

Let's illustrate how this technique works by opening and modifying the CONTACT form that comes with Visual dBASE. We'll add the codeblock

```
{;this.Print()}
```

to the CONTACT form's OnRightDblClick property. Then, when an operator double-clicks the right mouse button, dBASE will print the visible contents of that form.

You'll find the CONTACT file in the SAMPLES subdirectory. If you're using Visual dBASE under Windows 95, you click the Choose Directory folder and locate the SAMPLES subdirectory with the Choose Directory dialog box, as shown in **Figure A**.

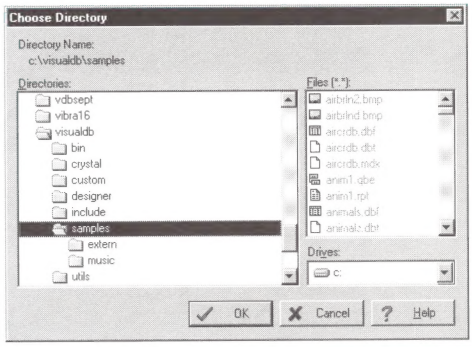
Modifying the form

Before you add the codeblock, you must open the form. You can do so with the File menu, or by double-clicking on the filename in the Navigator window. If the form doesn't open in design mode, select Design from the View menu. **Figure B** shows the CONTACT form in Design mode.

The Events tab

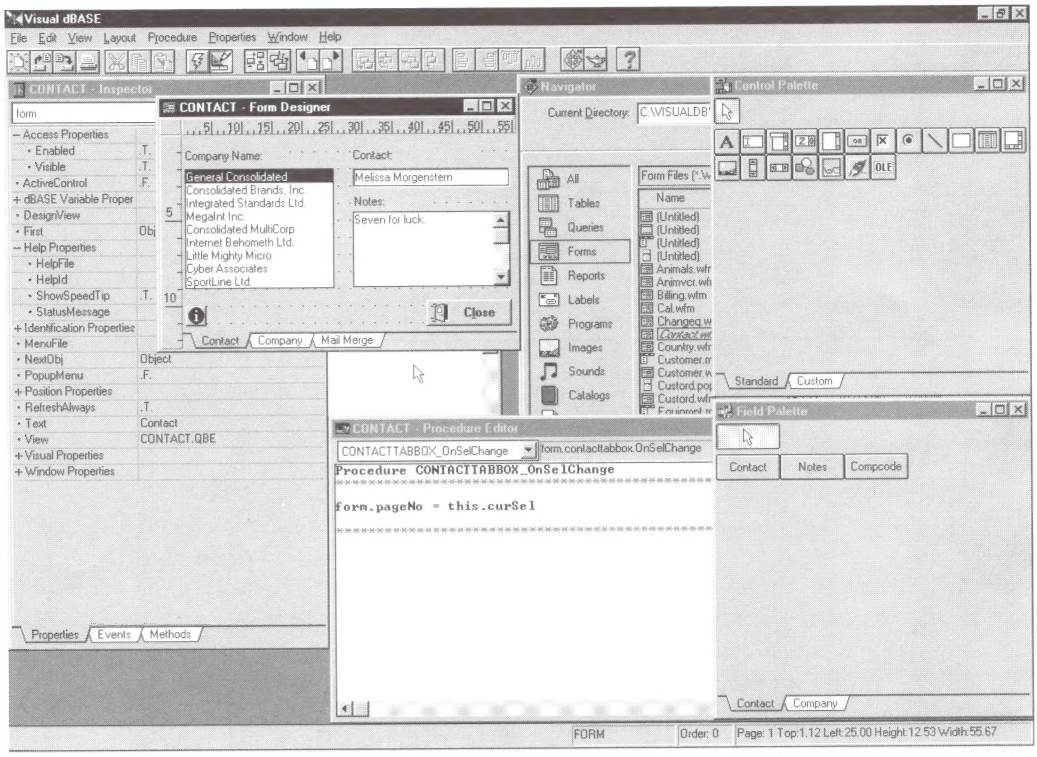
Next, click on the Inspector window and select the Events tab. You'll need this window to make your change to the form's properties. Instead of writing a program that leads the user through a process to print the form, you'll simply modify the Event properties, or the "what-happens-in-the-event-the-operator-takes-a-certain-action" properties of your form.

Figure A



You'll find our sample form in the SAMPLES subdirectory of your dBASE directory.

Figure B



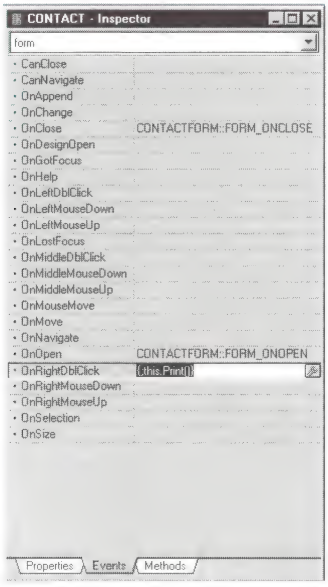
You need to view the form in Design mode before you can add the codeblock.

Now locate the OnRightDblClick property, which is the property that tells dBASE what to do when the user double-clicks an object or an area with the right mouse button. To open the text entry field, click once on the OnRightDblClick property. At this point, enter the codeblock

```
{;this.Print()}
```

as shown in Figure C. Be sure to enclose the codeblock within curly braces. With this change in effect, the end user can print the current form simply by double-clicking the right mouse button.

Figure C



Adding the option to print the form is as easy as entering a single instruction.

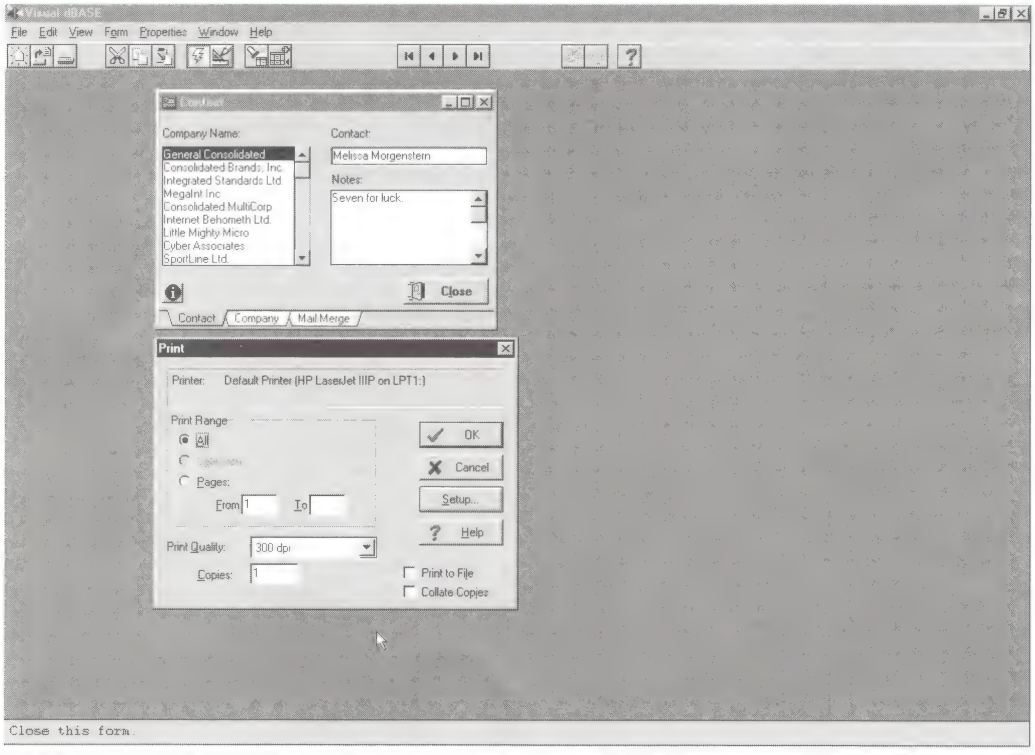
Testing the technique

You can test the change to the Events property by pressing [F2] to switch from Design mode, or Design view, to Form view. Place your cursor on some portion of the CONTACT form, away from any specific object, then double-click the right mouse button. When you do, dBASE will display a print selection box that lets you select the number of pages to print, print quality, number of copies, and printer output destination, as shown in Figure D.

Conclusion

Your end users must be able to print a form on demand. In this article, we showed you how to provide the ability to print a form by simply adding a single command to an Event property of a specific event—the OnRightDblClick event. You can apply this technique to assign almost any codeblock to a keyboard or mouse event in your applications. ❖

Figure D



Right-clicking on the form displays the Print dialog box.

Letting end users jump to DOS

Most Windows applications let the operator temporarily interrupt processing and open a window to perform DOS operations. In Visual dBASE, the DOS command lets you open a DOS window from within your application. In this article, we'll show you how to use this command from a pushbutton on a form.

The DOS command

You open a DOS window from within Visual dBASE by issuing the command *DOS*. When you place this command in your code, it interrupts program flow and displays the DOS prompt in a window. After you finish entering DOS commands, you simply type *EXIT* at the DOS prompt, and press [Enter]. When you do, the DOS window closes and your program resumes.

Creating a "Go to DOS" button

Suppose you want the operator to be able to open a DOS window by choosing an option from your menu. To provide this option, simply include the DOS command in a codeblock that you attach to the appropriate property of your menu choice.

For instance, you can attach the statement codeblock

```
{;DOS}
```

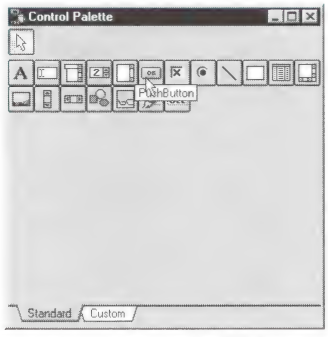
to the button's OnClick property. This command tells dBASE to execute the command in the codeblock each time the end user clicks that button.

One step at a time

For this example, let's create a blank form and add a simple "Go to DOS" pushbutton. To begin, open the File menu, choose New, and then select Form. When the New Form dialog box appears, click Designer. Visual dBASE will display the new form in Design mode. If the Control Palette isn't visible, right-click on the new form and choose Control Palette to display it. Then, click once on the pushbutton icon, as indicated in **Figure A**.

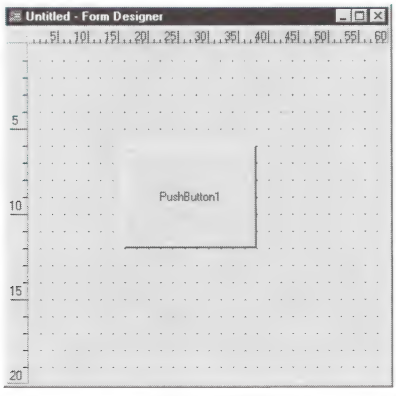
While holding down the left mouse button, drag the mouse to create a square the size you want the new button to be.

Figure A



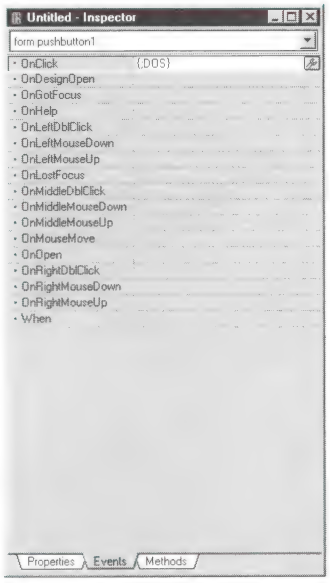
You can drag or double-click the pushbutton icon to place it on the new form.

Figure B



We'll use this pushbutton to open a DOS window from within our form.

Figure C



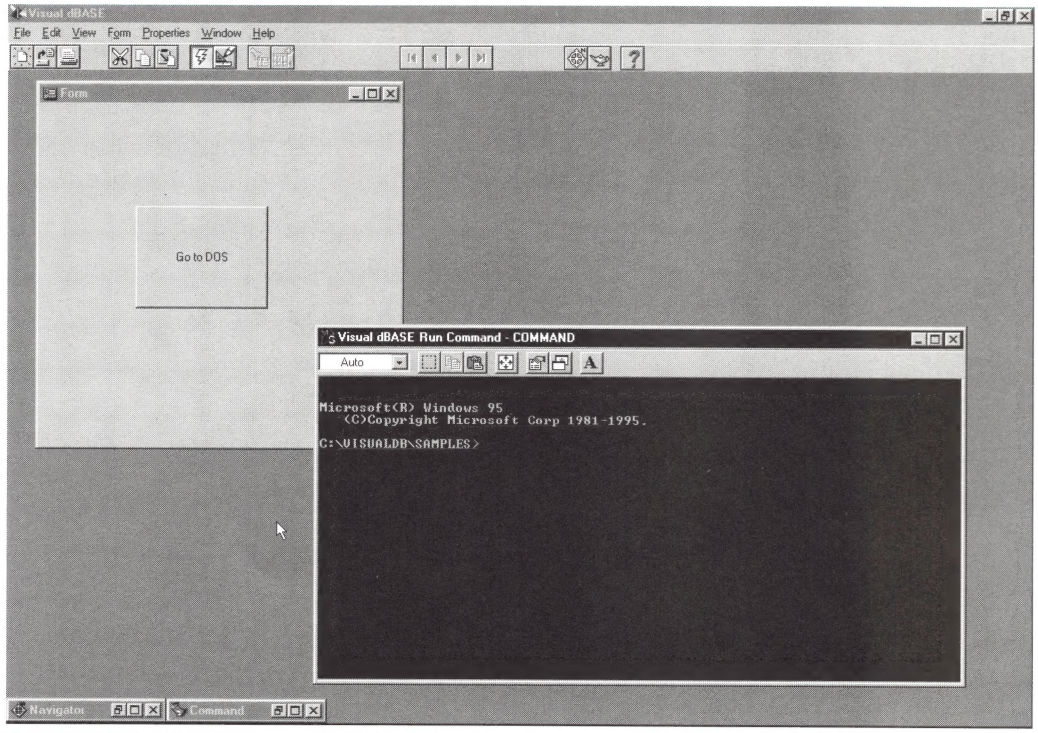
This screen shows a completed codeblock entry in the pushbutton's OnClick property.

Figure B shows the new pushbutton we added to our blank form.

Next, place the cursor over the new pushbutton and right-click. Then, choose Inspector to display the properties associated with this pushbutton object. Click the Events tab and locate the OnClick event at the top of the list. Add the command *{;DOS}* in the entry field for this event, as shown in Figure C.

Before you save your form, you may want to change the button's label from *PushButton1* to something more meaningful. To do so, click the Properties tab and change the entry in the Text property to *Go to DOS*.

Figure D



Making a Go to DOS button is easy!

Testing the new button

At last, you're ready to test your DOS button! To do so, click once on your form to highlight it, and then press [F2]. When you do, Visual dBASE will ask you to name your new form, then display it for your test. Click once on your new button to activate it. At this point, the DOS window will open and allow you to enter DOS commands, as shown in Figure D. The DOS window will stay open until you enter the *EXIT* command or until you click the window's close box.

Conclusion

In this article, we introduced the DOS command and showed you how to use it to open a DOS window from a pushbutton on a form. Our technique makes it easy for you to add a sophisticated option to applications you create with Visual dBASE. ❖

COMMAND SYNTAX

A new look at parameters

When you want to perform a block of code using earlier versions of dBASE, you create a procedure and then call it from your program. You'll be glad to know Visual dBASE has expanded the scope of the PROCEDURE command to enhance your ability to call blocks of code from within your programs. In this article, we'll examine the new features of this command and show you how to use them in your programs.

The PROCEDURE command

The enhanced format for the PROCEDURE command allows you to define parameters on the first line of code. The definition takes the form

```
PROCEDURE proc_name(parameter_list)
```

where you replace *proc_name* with the procedure name, and then, on the same line, you replace *parameter_list* with any passed

parameters. You must enclose these parameters in parentheses. This format differs from the earlier versions of dBASE in which the parameters reside on separate lines of code. There are also new naming conventions that you must keep in mind when you create your procedures.

For instance, in dBASE for DOS, you can have a procedure and a user-defined function (UDF) with the same name at the same time, because you call them separately. In Visual dBASE, if you declare a procedure and a UDF with the same name, the first one you declare is the only one the program recognizes. You can use up to 32 characters for the name of your Visual dBASE procedure—unlike dBASE for DOS, which recognizes only the first nine characters of your procedure name.

UNITED STATES POSTAL SERVICE™

Statement of Ownership, Management and Circulation
(Required by 39 U.S.C. 3685)

1. Publication Title	2. Publication number	3. Filing Date
Inside dBASE for Windows	0012689	10/3/96
4. Issue Frequency	5. No. of Issues Published Annually	6. Annual Subscription Price
Monthly	12	$79 ($89 Foreign)

7. Complete Mailing Address of Known Office of Publication (Not printer) (Street, City, County, State and ZIP+4 Code)
The Cobb Group
9420 Bunsen Parkway, Louisville KY 40220

8. Complete Mailing Address of the Headquarters of General Business Offices of the Publisher (Not printer)
The Cobb Group
9420 Bunsen Parkway, Louisville KY 40220

9. Full Names and Complete Mailing Address of Publisher, Editor, and Managing Editor (Do not leave blank)
Publisher (Name and Complete Mailing Address)
John Jenkins
The Cobb Group, 9420 Bunsen Parkway, Louisville KY 40220
Editor (Name and Complete Mailing Address)
Jeff Davis
The Cobb Group, 9420 Bunsen Parkway, Louisville KY 40220
Managing Editor (Name and Complete Mailing Address)
Michael Stephens
The Cobb Group, 9420 Bunsen Parkway, Louisville KY 40220

10. Owner (Do not leave blank. If the publication is owned by a corporation, give the name and address of the corporation immediately followed by the names and addresses of stockholders owning or holding 1 percent or more of tot amount of stock. If not owned by a corporation, give the the names and addresses of the individual owners. If owned by a partnership or other unincorporated firm, give its name and address, as well as that of each individual mu If the publication is published by a nonprofit oganization give its name and address.)

Full Name	Complete Mailing Address
Ziff Davis Publishing Company	1 Park Avenue, New York N.Y. 10016
Softbank Holdings Inc.	10 Langley Road, Suite 403 Newton Center, MA 02159

11. Known Bondholders, Mortgagees, and Other Security Holders Owning or Holding 1 Percent or More of Total Amount of Bonds, Mortgages or Other Securities. If there are none, check box ☑ None

Full Name	Complete Mailing Address

13. Title of Publication	14. Issue Date for Circulation Data Below
Inside dBASE for Windows	November 1996

15. Extent and Nature of Circulation	Average No. Copies Each Issue During Preceding 12 Months	Actual No. Copies of Single Issue Published Nearest to Filing Date
A. Total No. Copies *(Net Press Run)*	3,524	2,871
B. Paid and/or Requested Circulation		
1. Sales through dealers and carriers, street vendors and counter sales *(Not mailed)*	35	41
2. Paid or Requested Mail Subscriptions *(include advertiser's proof copies and exchange copies)*	2,811	2,014
C. Total Paid and/or Requested Circulation *(Sum of 15b(1) and 15b(2))*	2,845	2,055
D. Free Distribution by Mail *(Samples, complimentary, and other free)*	18	13
E. Free Distribution Outside the Mail *(Carriers or other means)*		
F. Total Free Distribution *(Sum of 15d and 15e)*	18	13
G. Total Distribution *(Sum of 15c and 15f)*	2,863	2,068
H. Copies Not Distributed		
1. Office use, left over,unaccounted, spoiled after printing	661	803
2. Return from News Agents	-	-
L. TOTAL *(Sum of 15g, 15h(1), and 15h(2))*	3,524	2,871
Percent Paid and/or Requested Circulation *(15c/15g x 100)*	99.37%	99.37%

This Statement of Ownership will be printed in the January issue of this publication.

Signature and Title of Editor, Publisher, Business Manager, or Owner — Date

Director-Fulfillment Operations

I certify that all information furnished on this form is true and complete. I understand that anyone who furnishes false or misleading information on this form or who omits material or information requested on the form may be subject to criminal sanctions (including fines and imprisonment) and/or civil sanctions (including multiple damages and civil penalties).

PS Form 3526, September 1995

Passing memory variables by reference

You can pass a variable to a procedure in two ways: by reference or by value. When you want the procedure to modify the value within the passed parameter, you must pass it by reference.

Let's take a look at a simple example of passing a value by reference:

```
sample = 10
DO add10 WITH sample
? sample  && This will display 20.

PROCEDURE add10(number)
number = number + 10
RETURN
```

When you pass a memory variable to a procedure by reference, the called procedure can change the value of the passed variable. In this example, the calling procedure changes the value assigned to the variable named *sample*.

Passing memory variables by value

The second method of passing a variable to a procedure is by value. You must use this method when you want the called procedure to have access only to the value contained in the variable. The called procedure cannot change the contents of the variable itself. Let's use the sample we used previously, but this time we'll pass the *sample* parameter as a value:

```
sample = 10
DO add10 WITH (sample)
? sample  && This will display 10.

PROCEDURE add10(number)
number = number + 10
RETURN
```

When you enclose the memory variable *sample* in parentheses, you pass it by value, not by reference. Therefore, the value doesn't change, even though the called procedure adds 10. When passing memory variables to a procedure, you must decide how you want the called procedure to act upon the variables you pass.

Returning values

In dBASE for DOS, you must define a function when you want a block of code to return a value to your calling program. In Visual dBASE, you can optionally return a value from a procedure by using the RETURN statement in the form

```
RETURN return_expression
```

where you replace *return_expression* with the expression, memory variable, or value you want to return to the calling program.

Let's create a sample procedure that returns a value. Suppose you want to check an employee record for a scheduled review date. If the review date in the record is past due, you want to display a message onscreen to notify the operator. You can code such a routine with the following commands:

```
IF review_check(employee.revdate)
  result= ;
  MSGBOX("Review scheduled for "+;
  DTOC(revdate))
ENDIF
```

```
PROCEDURE review_check(indate)
RETURN IIF(indate<DATE(),.T.,.F.)
```

When you call the procedure, it compares the passed date with the current date. If the passed date is less than the current date, the procedure returns a logical True; if not, it returns False. When the procedure returns True, the message box appears, notifying the operator of the past-due review date.

Some limitations and final thoughts

In dBASE for DOS, you could create a PRG file containing up to 963 procedures. You then would use the SET PROCEDURE TO command to make all the procedures available to your application. In Visual dBASE, you can store up to 193 procedures in a single PRG file. However, you can use the ADDITIVE option of the SET PROCEDURE TO command to allow your application to access additional procedure files as needed. In this way, your application has unlimited access to your custom procedures. ❖

INSIDE VISUAL dBASE™

Inside Visual dBASE (ISSN 1084-1970) is published monthly by The Cobb Group.

Staff:
Editor-in-Chief ..... Jeff E. Davis
Associate Editor-in-Chief ..... Tiffany M. Taylor
Publications Coordinator ..... Maureen Spencer
Editors ..... Laura Merrill
Joan McKim
Elisabeth Pehlke
Production Artist ..... Natalie Strange
Product Group Manager ..... Mike Stephens
Circulation Manager ..... Mike Schroeder
VP/Publisher ..... Mark Crane
President ..... John A. Jenkins

Address:
Please send tips, special requests, and other correspondence to
The Editor, *Inside Visual dBASE*
9420 Bunsen Parkway, Suite 300
Louisville, KY 40220
E-mail address: visual_dbase_win@merlin.cobb.zd.com

For subscriptions, fulfillment questions, and requests for group subscriptions, address your letters to
Customer Relations
9420 Bunsen Parkway, Suite 300
Louisville, KY 40220
E-mail address: customer_relations@merlin.cobb.zd.com

Phone:
Toll free US ..... (800) 223-8720
Toll free UK ..... (0800) 961897
Local ..... (502) 493-3300
Customer Relations Fax ..... (502) 491-8050
Editorial Department Fax ..... (502) 491-3433

Back Issues:
To order back issues, call Customer Relations at (800) 223-8720. Back issues cost $8.50 each, $8.95 outside the US. You can pay with MasterCard, VISA, Discover, or American Express, or we can bill you.

Postmaster:
Periodicals postage paid in Louisville, KY.
Postmaster: Send address changes to:
Inside Visual dBASE
P.O. Box 35160
Louisville, KY 40232

Prices:
Domestic ..... $79/yr ($8.50 each)
Outside US ..... $89/yr ($8.95 each)

PERIODICALS MAIL

Borland Technical Support
Technical Support
Information Line: (800) 523-7070
TechFAX System: (800) 822-4269

Please include account number from label with any correspondence.

PRINTING TIP

Changing orientation for printed pages

In "Letting Your Users Select a Printer," on page 4, we show you how to use the CHOOSEPRINTER( ) function within your application. That function lets your end users choose a printer or customize printer properties using the Print Setup dialog box. One of the properties the user can change with that dialog box is the page orientation—that is, portrait (normal, vertical page orientation) or landscape (horizontal page orientation).

Although CHOOSEPRINTER( ) lets the operator change the orientation, you may find it useful to change the orientation within your program. To do so, you simply change the value assigned to the system memory variable named _porientation.

The _porientation system memory variable can have one of two values—PORTRAIT or LANDSCAPE. You can assign its value by using a replacement statement in the form

```
_porientation = "LANDSCAPE"
```

This page orientation option gives you lots of flexibility in designing your report-processing programs.

For instance, suppose you have a report that requires the landscape orientation, but you want to give the operator the ability to select which printer to use to print the report. You can combine the page orientation setting with the CHOOSEPRINTER( ) function by coding a routine like this:

```
IF CHOOSEPRINTER( )
 _porientation="LANDSCAPE"
 REPORT FORM myreport TO PRINT
ENDIF
```

Although the end user could change the orientation when CHOOSEPRINTER( ) displays the Print Setup dialog box, the command

```
_porientation="LANDSCAPE"
```

ensures that the orientation is set to the landscape option.

Customizing the settings of Visual dBASE system memory variables provides an excellent way to control the environment from within your program. The page orientation variable is just one of more than 20 print-related variables you can modify to customize your printed reports. ❖

Printed in USA
This journal is printed on recyclable paper.